A FACTOR GRAPH APPROACH TO CONSTRAINED OPTIMIZATION

A Thesis
Presented to
The Academic Faculty

By

Ivan Dario Jimenez

In Partial Fulfillment
of the Requirements for the Degree
B.S. in Computer Science with the Research Option
in the College of Computing

Georgia Institute of Technology

December 2016

A FACTOR GRAPH APPROACH TO CONSTRAINED OPTIMIZATION

Approved by:

Dr.Frank Dellaert
School of Computer Science,
College of Computing
Georgia Institute of Technology

Dr.Byron Boots
School of Computer Science,
College of Computing
Georgia Institute of Technology

Date Approved: December 15, 2016

ACKNOWLEDGEMENTS

I would like to especially thank Doctor Duy-Nguyen Ta and Jing Dong for their invaluable help and guidance throughout the research process. Without them this thesis would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Several problems in robotics can be solved using constrained optimization. For example, solutions in areas like control and planning frequently use it. Meanwhile, the Georgia Tech Smoothing and Mapping (GTSAM) toolbox provides a straight forward way to represent sparse least-square optimization problems as factor graphs. Factor graphs, are a popular graphical model to represent a factorization of a probability distribution allowing for efficient computations. This paper demonstrates the use of the GTSAM and factor graphs to solve linear and quadratic constrained optimization programs using the active set method. It also includes an implementation of a line search method for sequential quadratic programming that can solve nonlinear equality constrained problems. The result is a constrained optimization framework that allows the user to think of optimization problems as solving a series of factor graphs and is open-source.

CHAPTER 1
INTRODUCTION AND BACKGROUND

Factor graphs are probabilistic graphical models that represent random variables and an unconstrained objective function between them. They are a way to express problems in Simultaneous Localization and Mapping (SLAM) [1] and Model Predictive Control (MPC) [2]. As you can see in figure 1.1, these are bipartite graphs where variable nodes represent unknown random variables and factor nodes represent evidence on those variables gathered from measurements or prior knowledge.

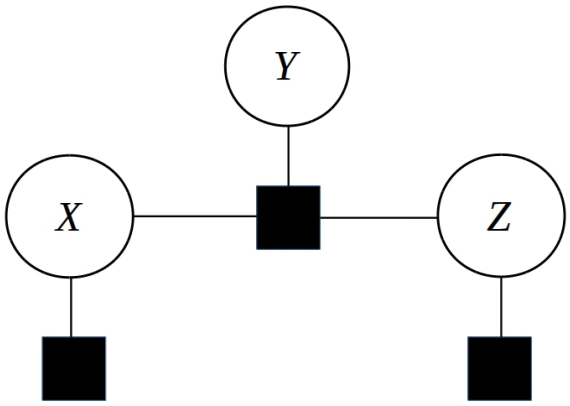

Figure 1.1: A sample factor graph on three variables represented by the circle nodes labeled X, Y and Z. The black boxes represent factors: evidence on the value of the variables that they connect to.

Georgia Tech Smoothing and Mapping (GTSAM) Toolbox is frequently used to solve factor graphs. GTSAM exploits the sparse factor graph representation of a problem to solve it as a sum of least squares as shown in Equation 1.1. It is important to note here that this limits the scope of factor graph problems to unconstrained optimization problems. Although there are methods to solve constrained optimization problems with unconstrained optimization methods, they have the drawback of having

to manually set the penalty parameters [3].

$$\min \sum_{i}^{n} ||r_i(x)||_{\Sigma}^2 \tag{1.1}$$

The goal of this research is to solve constrained optimization problems using a series of factor graphs. Specifically, this research focuses on inequality constrained optimization problems that take the form of Equation 1.2. In this form, f is the objective function, $\mathcal{E}$ is the set of equality constraint indexes and $\mathcal{I}$ the set of inequality constraint indexes. If the objective function f or the constraints c are nonlinear, there might be many or no solutions to the problem. Thus, the scope of the problem is restricted to convex feasible regions where a unique solution can be found.

$$\begin{aligned} &\min_x f(x) \\ \text{s.t.} \quad &c_i(x) = 0 \quad i \in \mathcal{E} \\ &c_i(x) \leq 0 \quad i \in \mathcal{I} \end{aligned} \tag{1.2}$$

Linear Programs constitute the basis of a hierarchy of constrained optimization problems where solvers for problems of higher complexity require solvers for programs of lower complexity. In order to solve quadratic constrained optimization problems of the form Equation 1.4, a solver has to be able to solve Linear Programs which take form Equation 1.3. Linear programming algorithms are a mature technology that can manage even large versions of these problems.

$$\begin{aligned} &\min_x c^T x \\ \text{s.t.} \quad &A_i x = 0 \quad i \in \mathcal{E} \\ &A_i x \leq 0 \quad i \in \mathcal{I} \end{aligned} \tag{1.3}$$

In a similar way to linear programming, a quadratic programming solver is required to solve nonlinear programs. Quadratic programs are the same type as Equation 1.2 but they take the form of Equation 1.4. Unlike linear programs, however, not all quadratic programs are guaranteed to be convex if the feasible region is convex. Thus, the scope of this paper only includes quadratic programs where the matrix H is positive definite.

$$\begin{aligned} & \min_x \frac{1}{2} x^T H x - x^T g + \frac{1}{2} f \\ \text{s.t.} \quad & A_i x = 0 \quad i \in \mathcal{E} \\ & A_i x \leq 0 \quad i \in \mathcal{I} \end{aligned} \tag{1.4}$$

The ultimate goal is to use factor graphs to solve nonlinear programs that arise in areas of robotics such as Task Plan Optimization [4], Model Predictive Control [5] and SLAM [6]. This research provides one of the few open-source alternatives to proprietary solver libraries such as SNOPT [7]. It also does so in the expressive language of factor graphs. Combined, these factors will enable GTSAM to solve a wide variety of problems in robotics.

CHAPTER 2

METHODS

2.1 Active Set Method for Linear Programming: The Simplex Method

2.1.1 Overview

This is an implementation of algorithm 16.3 from [3] that solves problems of the form shown in Equation 1.3. The intuition behind iterative optimization algorithms is to improve on a given solution by solving a simpler problem. Specifically, the simpler problem in the case of Active Set Methods can be interpreted as sliding along the vertexes of the polytope created by the inequality constraints of the problem in the direction that minimizes the gradient of the cost function as seen in figure 2.1.

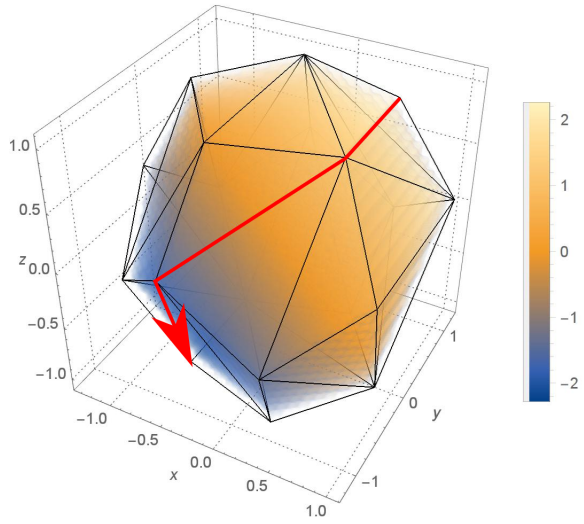

Figure 2.1: This is a visual representation of what a the active set method does when solving $\min x + y + z$ subject to the inequalities that define the polytope shown.

Algorithm 1 The Active Set Method for Linear Programming

procedure ACTIVE-SET SOLVER($x_0, \lambda_0, f, c, \mathcal{I}, \mathcal{E}$)
  $\mathcal{W} \leftarrow \{c_i | i \in \mathcal{E}\} \cup \{c_i | c_i(x_0) = 0 \wedge i \in \mathcal{I}\}$
  $x_k \leftarrow x_0$
  $\lambda_k \leftarrow \lambda_0$
  $k \leftarrow 1$
  $converged \leftarrow false$
  while !$converged$ **do**
    $\mathcal{WG} \leftarrow BuildWorkingGraphLP(\mathcal{W}, x_k)$
    $k \leftarrow k + 1$
    $x_k \leftarrow \mathcal{WG}.optimize()$
    $p_k \leftarrow x_k - x_{k-1}$
    if $p_k = 0$ **then**
      $\mathcal{DG} \leftarrow BuildDualGraph(f, c, x_k)$
      $\lambda_k \leftarrow \mathcal{DG}.optimize()$
      $leavingFactor \leftarrow \{C_i | \lambda_i > 0 \wedge i \in \mathcal{W}\}$
      if $leavingFactor = \emptyset$ **then**
        $converged \leftarrow true$
      else
        $\mathcal{W} \leftarrow \mathcal{W} - leavingFactor$
    else
      $(\alpha, factor) \leftarrow computeStepSizeLP(x_{k-1}, p_k, \mathcal{W})$
      if $factor! = \emptyset$ **then**
        $\mathcal{W} \leftarrow \mathcal{W} \cup factor$
      $x_k \leftarrow x_{k-1} + \alpha p_k$
  return (x_k, λ_k)

2.1.2 Algorithm

First, the starting iterate and the feasible region of the program are used to create the first working set. The iterate is defined by a start point x_0 and initial λ values λ_0. The feasible region is implicitly defined by the set of constraints c and the indexes of equality and inequality constraints, $\mathcal{E}$ and $\mathcal{I}$ respectively. From this information an active set is selected: the union of all equality constraints and all inequalities such that $c_i(x) = 0, i \in \mathcal{I}$. With the initialization complete, the first iteration can begin.

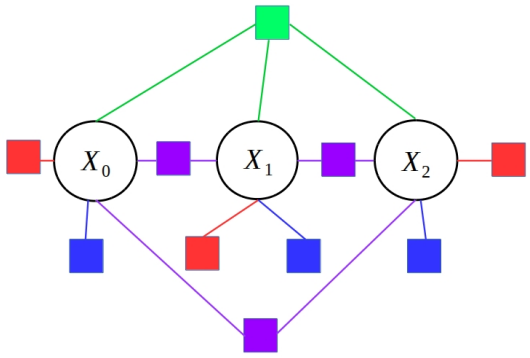

Figure 2.2: This a visual representation of a working graph on three variables. The red nodes represent the unary factor for $||x - x_k - (-\nabla_x f^T x)||^2_\Sigma$. The blue, purple and green factors represent unary, binary and ternary constraints on the three variables of the form $||c_i(x)||^2_\Sigma$ where $c_i(x) = Ax - b, i \in \mathcal{I} \cup \mathcal{E}$

.

After initialization, a working graph $\mathcal{WG}$ must be constructed. This factor graph, shown in figure 2.2, finds a local minimum on the part of the constraint surface defined by the active set $\mathcal{W}$. Notice that the graph in that figure is a worst case scenario where there are constraints on all subsets of variables. In reality, constraints are only present for some subsets of variables which justifies a sparse representation. Clearly, the factors with the form $||c_i(x)||^2_\Sigma$ are there to ensure the result stays within the constraint surface. On the other hand, the unary factor $||x - x_k - (-\nabla_x f^T x)||^2_\Sigma$ tries to match the step $x - x_k$ with the negative gradient of the cost function $-\nabla_x f^T x$.

The result of optimizing this graph is an iterate x which would result from moving from x_k in the direction of the negative gradient of the cost function while staying in the constraint surface.

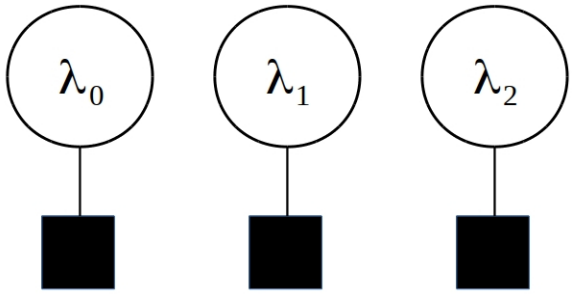

Figure 2.3: The dual graph that solves for the lagrange multipliers of a linear program. Each factor has the form $||\nabla_x f - \lambda_i \nabla_x c_i||^2_\Sigma$ where $i \in \mathcal{I}$. This dual graph in particular is a special case of a problem with three variables each with one unary constraint.

If the result of optimizing the factor graph is the same as the starting point, the algorithm has reached a local minimum and must either converge or change the active set. More specifically, if the result is not the global solution, the algorithm removes a constraint from the current active set that is blocking its advance. In order to tell which one is the blocking constraint, a dual graph is built to get the lagrange multipliers for each constraint. The Dual Graph shown in figure 2.2 is equivalent to optimizing the lagragian: $\nabla_x \mathcal{L}(x, \lambda) = 0$ where $\mathcal{L}(x, \lambda) = f(x) - \lambda c(x)$.

Generating the dual graph can be thought of as a transformation on the factor graph of the constraints. In this transformation each constraint factor is replaced by its corresponding dual variable λ. Each variable node is replaced by a factor corresponding to the gradient of the lagrangian with respect to that variable. Algorithm 2 shows this process.

Algorithm 2 Building the Dual Graph

```
1: procedure BuildDualGraph(f, c, x_k)
2:     DG ← ∅
3:     for v ∈ VARS(c) do
4:         DG ← DG ∪ {||∇_v cλ − ∇_v f||^2}
       return DG
```

The next step is to interpret the lagrange multipliers. If λ_i for some constraint $c_i,\ i \in \mathcal{I}$ is positive on the current point x, it is understood that the constraint c_i must be the leaving constraint and can removed safely. Furthermore, a zero λ implies the minimum is on the constraint while a negative λ implies removing the constraint from the active set would cause the iterate to leave the feasible region.

Algorithm 3 Computing step size α and *leavingFactor* for linear programs.

```
1: procedure COMPUTESTEPSIZELP(x_{k-1}, p_k, W)
2:     α ← ∞
3:     leavingFactor ← ∅
4:     for {c_i | i ∈ I ∧ c_i ∉ W} do
5:         if A_i^T p_k > 0 then
6:             α_i ← (b − A^T x_{k-1}) / (A^T p_k)
7:             if α_i < α then
8:                 α ← α_i
9:                 leavingFactor ← C_i
       return (α, leavingFactor)
```

Meanwhile, if the result of the working graph optimization is different from the current solution, the iterate can advance in the direction $p_k = x_k - x_{k-1}$. The working set only operates in a subset of the problem's constraints. Thus, the step size p_k may need to be scaled by a factor α before continuing such that the resulting iterate doesn't violate any of the other constraints. If α is used to put the resulting iterate on a constraint, that constraint has to be added to $\mathcal{W}$. More generally, you can think of computing a step size as a line search in the direction p_k.

$$A_i x_{k-1} - b_i \leq 0 \quad \forall_i\ i \in \mathcal{I} \tag{2.1}$$

$$A_i(x_{k-1} + \alpha p_k) - b_i \leq 0 \quad \forall_i\ i \in \mathcal{I} - \mathcal{W} \tag{2.2}$$

$$A_i x_{k-1} + A_i \alpha p_k \leq b_i \quad \forall_i\ i \in \mathcal{I} - \mathcal{W} \tag{2.3}$$

$$\alpha = \frac{b_i - A_i x_{k-1}}{A_i p_k} \quad \forall_i\ i \in \mathcal{I} - \mathcal{W} \tag{2.4}$$

Computing α warrants further explanation. Its derivation starts from the assumption that x_{k-1} does not violate any constraint which can be represented as Equation 2.1. The goal is to ensure that the same holds true for the updated iterate as well which is shown in Equation 2.2. From Equation 2.3 it is clear that if $A_i \alpha p_k \leq 0$ the constraint is not violated by the current scaled step size. Since the step size should be as big as possible, α is solved for using the inequality shown in Equation 2.3 which results in Equation 2.4.

2.1.3 Initialization

$$\begin{aligned} & \min_{\ell} \ell \\ \text{s.t.} \quad & A_i x + \ell = 0 \quad i \in \mathcal{E} \\ & A_i x + \ell \leq 0 \quad i \in \mathcal{I} \end{aligned} \tag{2.5}$$

Initializing the solver without an a given feasible initial value requires solving a different linear program which changes the objective function to a slack variable ℓ and adds that same slack variable to the constraints as shown in Equation 2.5. The intuition behind this is that minimizing the slack (or violation of the constraints) will yield a feasible point that can be used by the algorithm. Notice that any value for the original set of variables will lead to a feasible point in the new problem.

2.2 Active Set Method for Quadratic Programming

2.2.1 Overview

Quadratic programs can be solved in a similar fashion to Linear Programs with one added caveat: as shown in figure 2.4, an optimal solution may not necessarily fall on a vertex of the constraint surface. Thus, to the algorithm is modified slightly to ensure convergence to the right location.

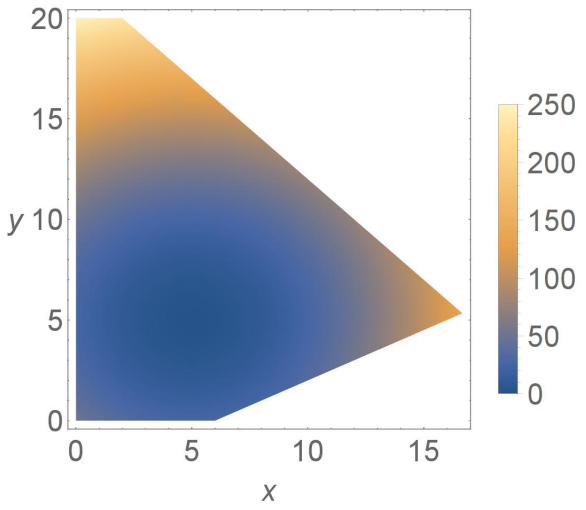

Figure 2.4: This is a graph of the feasible region for optimization problem with cost function $x^2 + y^2 - 10y - 10x + 50$ subject to a set of linear constraints.

2.2.2 Algorithm

Building $\mathcal{WG}$

The Working Graph $\mathcal{WG}$ changes to include the objective function as a factor to minimize instead of $||x - x_k - (-\nabla_x f^T x)||^2_\Sigma$. This takes advantage of the fact that GTSAM can directly optimize quadratic factors like those used to express the objective function in a quadratic program.

Computing α

Unlike linear programs where α could be greater that 1, step sizes in quadratic programs are guaranteed to be less than the initial estimate. This comes from the fact that linear objective functions are monotonically decreasing in the direction $-\nabla_x f$. Meanwhile, quadratic objective functions lose monotonicity meaning that a solution to $\mathcal{WG}$ must the at minimum on the active set's constraint surface. Thus, α can be initialized to 1 instead of ∞.

2.2.3 Initialization

Since linear constraints are also being used, the same method for initializing linear programs can be used to initialize quadratic programs.

2.2.4 Implementation

Evidently, the method for solving quadratic and linear programming is almost identical. Thus, the GTSAM implementation of these algorithms is a single templated class that takes the differing functions as template parameters. More specifically, GTSAM uses the exact same code to solve quadratic and linear programs except for the generation of the working graph and computing α.

2.3 Line Search Sequential Quadratic Programming for Nonlinear Constrained Optimization

$$\begin{aligned} &\min_x f(x) \\ &\text{s.t.} \quad c(x) = 0 \quad i \in \mathcal{E} \\ &\qquad c(x) <= 0 \quad i \in \mathcal{I} \end{aligned} \tag{2.6}$$

This section presents an implementation of algorithm 18.3 from [3] that solves problems of the form shown in Equation 2.6 based on the algorithms presented previously. This algorithm belongs to the Line Search Family where it uses the QP sub-problem at each iteration to determine a direction and line search to determine the magnitude of the step. Notice, that this this algorithm solves an inequality constrained quadratic problem at each iteration.

2.3.1 Algorithm

Line Search Sequential Quadratic Programming starts by solving the inequality constrained quadratic program shown in line 10 of Algorithm 4. The results are p_k and $\widehat{\lambda_k}$ which represent the raw step direction and a new dual value used to calculate the direction of the dual step. With the information given by solving the inequality constrained quadratic program, we begin the line search at line 16.

The line search is ensuring that there is sufficient decrease in the merit function to justify the step. The quadratic program gives the algorithm knowledge of the direction in that will give progress but due to the approximate nature of linearization of constraints (especially inequality constraints), the step may need to be scaled down to ensure actual progress. The merit function attempts to balance the two often conflicting goals of ensuring the feasibility of the constraints and minimizing the error function.

The condition that satisfies the line search is comparing the value of the merit function ϕ_1 at the result of taking the step scaled by α and the sum of the current value of the merit function with the directional derivative of ϕ_1 in the direction of p_k, $D\phi_1$, scaled by α and a factor η.

After the line search has successfully concluded, the next iterate is defined by the current iterate plus the step determined by the quadratic program solved scaled by the alpha determined during the line search.

2.3.2 Merit Function

$$\phi_1(x;\mu) := f(x) + \mu||c(x)||_1 \tag{2.7}$$

In this implementation I have used Equation 2.7 where it is assumed all inequalities are treated as equalities with a slack variable that is not taken into account by the

Algorithm 4 The Line Search Sequential Quadratic Programming for Nonlinear Programs

procedure LINESEARCHSQP SOLVER($x_0, \lambda_0, f, c, \mathcal{I}, \mathcal{E}$)
  $\mu_k \leftarrow 0$
  $\rho \leftarrow 0.7$
  $\eta \leftarrow 0.3$
  $x_k \leftarrow x_0$
  $\lambda_k \leftarrow \lambda_0$
  $k \leftarrow 1$
  while $!checkConvergence(f, c, \mathcal{I}, \mathcal{E}, x_k, \lambda_k)$ **do**
    $k \leftarrow k + 1$
    $(p_k, \widehat{\lambda_k}) \leftarrow \arg\min_p p^T \nabla^2 \mathcal{L} p + p^T \nabla f_k$
      s.t. $\nabla c_i(x_k)p + c_i(x_k) = 0 \qquad i \in \mathcal{E}$
      $\nabla c_i(x_k)p + c_i(x_k) = 0 \qquad i \in \mathcal{I}$
    $p_\lambda \leftarrow \widehat{\lambda_k} - \lambda_k$
    $\mu \leftarrow \frac{\frac{1}{2} p_k^T \nabla^2 \mathcal{L} p_k + \nabla f_k p_k}{(1-\rho)||c(x_k)||_1}$
    if $\mu_k < \mu$ **then**
      $\mu_k \leftarrow \mu$
    $\alpha \leftarrow 1$
    while $\phi_1(x_k + \alpha p_k; \mu_k) > \phi_1(x_k) + \eta \alpha D\phi_1(x_k; \mu_k)$ **do**
      $\alpha \leftarrow \rho\alpha$
      if $\alpha < 10\rho$ **then**
        FAILURE TO CONVERGE
    $x_{k+1} \leftarrow x_k + \alpha p_k$
    $\lambda_{k+1} \leftarrow \lambda_k + \alpha p_\lambda$
  return (x_k, λ_k)

merit function. Although this function is not differentiable, it does have a directional derivative .

$$D_{p_k}\phi_1(x,\mu) := \nabla f(x)p_k - \mu||c(x)||_1 \tag{2.8}$$

From inspection it clear that μ is a multiplier on the violation on the constraints. Clearly, it is being used to balance the need to optimize the f with the need to avoid a violation of the constraints.

2.3.3 Checking for Convergence

The function *checkConvergence* must check that the current solution is feasible and that it satisfies the KKT conditions for optimality. The conditions can be divided

Algorithm 5 Procedure for checking convergence of the algorithm

1: **procedure** $CheckConvernce(f, c, \mathcal{I}, \mathcal{E}, x_k, \lambda_k)$
return $\forall_{i\in\mathcal{E}}||c_i(x)||_\infty < 0 \wedge$
$\forall_{i\in\mathcal{I}}\ c_i(x) \leq 0 \wedge$
$\nabla\mathcal{L}(x) = 0 \wedge$
$\forall_{i\in\mathcal{I}\cup\mathcal{E}}\lambda_i \geq 0$

into four clauses in a single and statement shown in Algorithm 5. The first and second clauses ensure the feasibility of the solution. The third clause ensures the stationary of the Lagrangian. Finally, the fourth clause ensures the dual feasibility of the KKT system.

2.4 Factor Graphs

There are two factor graphs that are used in the process of solving the nonlinear program: The factor graph used to represent the quadratic problem solved at each iteration and the dual graph used to determine whether the KKT conditions have been satisfied.

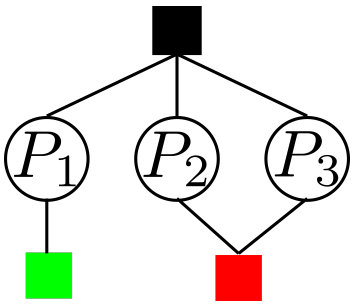

Figure 2.5: This factor graph is used to represent the linearized Quadratic Program Solved at each iteration. The black factor represents the quadratic cost function: $p^T \nabla^2 \mathcal{L} p + p^T \nabla f$. The green factor represents the error on the linearized equality constraints: $||\nabla^2 c_i(X)||_\Sigma^2 \quad | \quad i \in \mathcal{E}$. The red factor represents the error on the inequality constraints $||\max(0, \nabla^2 c_i(x))||_\Sigma^2 \quad | \quad i \in \mathcal{I}$.

The first factor graph is used to represent the quadratic program solved at each iteration. It is show in figure 2.5. Notice that there is one green green factor for each equality constraint and one red factor for each inquality constraint. This graph is then given to the quadratic solver.

The dual graph is the same as the one seen on figure 2.3. Both the variables and the factors hold the same meaning although the Lagrangian is now nonlinear. The main difference comes from how the factor is used. Whereas before the factor was optimized to obtain the λ values in QP and LP, during SQP the factor graph is given the dual variables in order to determine whether the $\nabla \mathcal{L} = 0$ necessary optimality condition is satisfied.

2.4.1 Limitations

Unlike previous iterations, this implementation requires a feasible starting point. The sequential quadratic programming solver provided can only solve problems where the hessian is positive semi-definite. A problem like those used in the previous solvers where a slack value is used to initialize the program is almost guaranteed to have a negative-definite matrix. That is because the hessian of the sub-problem is the hessian of the Lagrangian of the greater problem: $\nabla^2 \mathcal{L} = \nabla^2 f - \lambda \nabla c$. Unless the

constraints are linear, the hessian of the Lagrangian of Equation 2.9 is guaranteed to be negative if the hessian of the constraints is positive-definite.

$$\begin{aligned} &\min_{x} s \\ \text{s.t.} \quad & c(x) - s = 0 \quad i \in \mathcal{E} \\ & c(x) - s <= 0 \quad i \in \mathcal{I} \end{aligned} \tag{2.9}$$

In addition to this, the linear and quadratic approximations of the nonlinear functions become increasingly worse as the function grows faster. The algorithm thus, will fail to converge in very fast-changing functions such as e^x or $sin(x)$ but converges on low degree polynomials.

CHAPTER 3

RESULTS

3.1 Overview

This section will show how GTSAM goes about solving a set of constrained optimization problems. Both the quadratic and linear programs presented will have an arbitrary starting point $(2, 20)$ and will share the same set of inequality constraints.

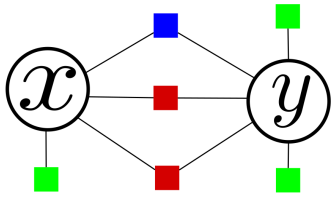

Figure 3.1: This factor graph can be used to solve both the linear and quadratic programs solved in Equation 3.1 and Equation 3.2 respectively. The blue factor can represent the objective function, the red factors represent the binary constraints on the variables and the green factors represent unary constraints on the variables.

Figure 3.1 shows the factor graph that can be used to represent both the linear and quadratic programs solved in this section. The green and red factors are used to represent the constraints on the variables. The blue factor can represent either the linear or quadratic objective functions. Notice that this factor is not solved directly. Instead, different subsets of the factors are selected in each iteration as part of the active set which eventually converges to the solution of the problem.

3.2 GTSAM Solving a Linear Program

$$
\begin{aligned}
&\min_x x^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
\text{s.t.} \quad & x \geq 0 \\
& y \geq 0 \\
& y \leq 20 \\
& y \leq -x + 22 \\
& y \geq 0.5x - 3
\end{aligned} \tag{3.1}
$$

The linear program presented in Equation 3.1 can have its constraints represented by figure 3.2. Clearly, this gives us a convex feasible region. Furthermore, upon visual inspection, it is obvious from figure 3.3 that the minimum inside this feasible region lies at the origin. A good optimization algorithm, however, must be able to converge from distant locations. To demonstrate this, the algorithm is initialized to $(2, 20)$. Although this happens to be a vertex between two constraints it does not need to be.

Upon initialization, Algorithm 1 must determine the active set for the given iterate. Figure 3.4 shows the state of the program upon determining that subset of the inequalities. Since the iterate is at the intersection of two inequalities, further optimization should not yield improvements (at least not in two dimensions). Thus, the next iteration must remove one of the active constraints. For that purpose the dual factor graph shown in figure 3.5 is built.

The dual factor graph is used as a way to solve $\nabla_x \mathcal{L}(x, \lambda) = 0$ to obtain the lagrange multipliers for the constraints in the active set. λ_4 corresponds to the constraint $y \leq -x + 22$ while λ_3 corresponds to $y \leq 20$. Optimizing this factor graph results in $(\lambda_4, \lambda_3) = (1, 0)$ which means that $y \leq -x + 22$, having the maximum

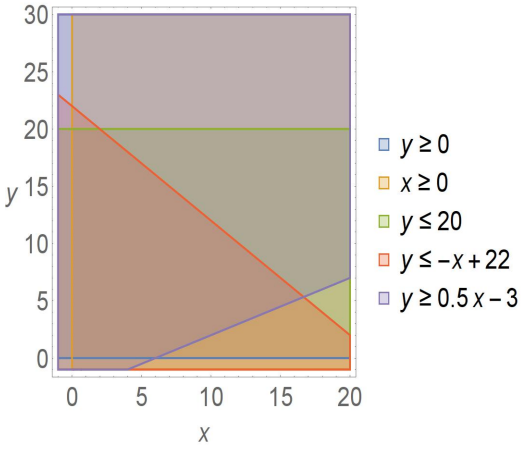

Figure 3.2: This is a graph of the feasible region for optimization problem shown in Equation 3.1 .

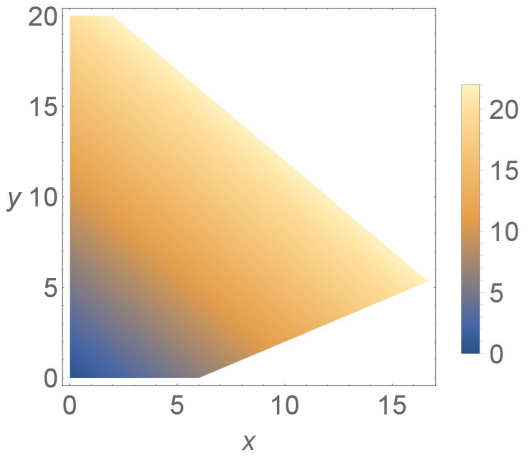

Figure 3.3: This is a graph of the feasible region with the cost function overlaid for optimization problem shown in Equation 3.1.

lagrange multiplier of the two should be removed from the active set. To understand the intuition behind this inspect figure 3.4. You can see that if you projected the gradient of the cost function unto the normal vector of both constraints, the constraint removed would have the longest projection.

The algorithm begins the second iteration with only constraint $y \leq 20$ in $\mathcal{W}$ and thus the iterate must move. The optimization of $\mathcal{WG}$ yields $x_k = (1, 22)$. Notice that this is not on the vertex of the polytope and isn't actually the next iterate. Instead, the vector $p_k = x_k - x_{k-1} = (-1, 0)$ defines the direction in which to search for the next iterate. Algorithm 3 returns an α of 2. Notice that this α can take values greater than one to ensure every iterate is at a vertex of the polytope by the end of the iteration.

Finally, the variable x_k can be assigned the final value of the iterate $x_{k-1} + \alpha * p_k = (0, 20)$. This means the iterate has entered in contact with constraint $x \geq 0$ and thus has to be added to the active set. This results in the state shown in figure 3.6.

Since there are two intersecting constraints in the working set, further optimization

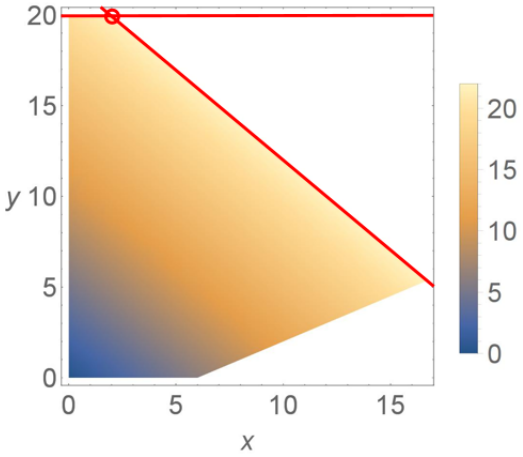

Figure 3.4: The starting iterate of the linear program in Equation 3.1. It starts at coordinates $(2, 20)$, marked with a red circle. The inequalities in the active set are highlighted in red.

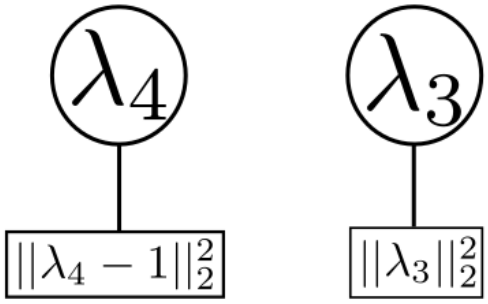

Figure 3.5: When solving Equation 3.2, this is the dual graph solved in the first iteration.

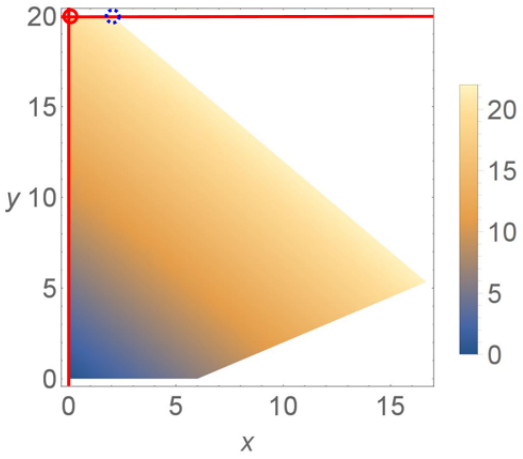

Figure 3.6: This is the second iteration of the Active Set Solver on Equation 3.1. The current iterate moved from the dashed blue circle at $(2, 20)$ to the location of the red circle at $(0, 20)$. The active set is highlighted in red.

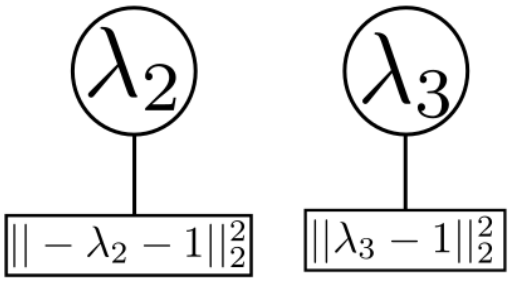

Figure 3.7: This is the dual graph solved by the Active Set Solver when solving Equation 3.1 on the third iteration.

of $\mathcal{WG}$ will not change the iterate. Instead the factor graph shown in figure 3.9 is used to choose a constraint to deactivate. From inspection of the graph, one can see

that only constraint $y \leq 20$ is stopping the optimization from following the gradient of the cost function. As expected, it has a dual variable λ_3 of 1 which is greater than λ_2 with a value of -1 for $x \geq 0$. Thus the constraint from $\mathcal{W}$ with the greatest lambda is removed and the loop continues.

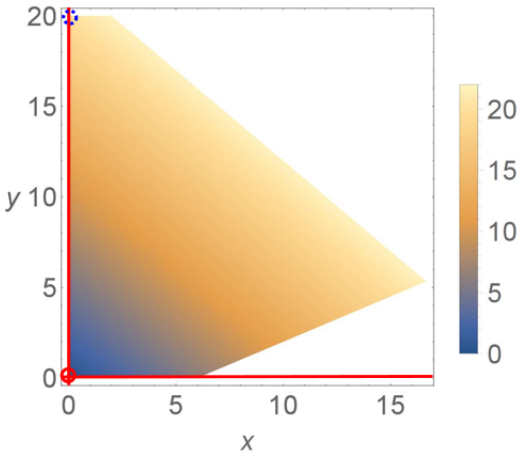

Figure 3.8: On the fourth iteration of solving Equation 3.1, the active set solver moves the current iterate from the location highlighted by the dashed blue circle at $(0, 20)$ to the red circle $(0, 0)$. The active constraints at the new iterate are highlighted in red.

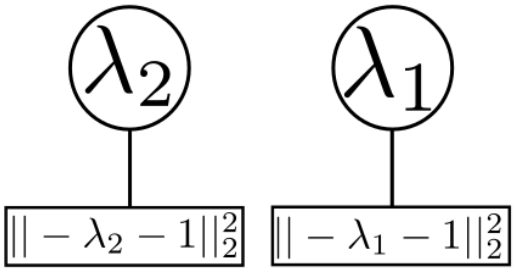

Figure 3.9: This is the dual graph solved during the fifth and final iteration of the active set solver when processing Equation 3.1.

In fourth iteration of the active set solver, the optimization of the working graph yields $x_k = (0, 19)$. Since it is a different from the current iterate a direction $p_k = (0, -1)$ is defined. With an α of 20, the iterate arrives at the origin: the minimum for this problem. Still, in this iteration x_k has collided with the constraint $y \geq 0$ and thus must be added to the working set.

In the final iteration of the algorithm, it checks for the Lagrange multipliers of the two constraints active at the origin by creating the dual graph shown in figure 3.9. Both of the multipliers are negative which pass the convergence condition and thus terminate the algorithm.

3.3 GTSAM Solving a Quadratic Program

$$\begin{aligned} \min_x \; & x^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - x^T \begin{bmatrix} 10 \\ 10 \end{bmatrix} + 50 \\ \text{s.t.} \quad & x \geq 0 \\ & y \geq 0 \\ & y \leq 20 \\ & y \leq -x + 22 \\ & y \geq 0.5x - 3 \end{aligned} \tag{3.2}$$

The quadratic solver concludes its initialization by determining the active set $\mathcal{W}$ highlighted in figure 3.10. As expected, the optimization on the working graph $\mathcal{WG}$ will yield no improvement on the current iterate. The algorithm then builds the dual graph shown in figure 3.11 which solves for the lagrange multipliers for the active constraints. Since the constraints are linear the process of building and solving this dual graph is exactly the same as in the linear programming version. Constraint $y \leq 20$ and $y \leq -x + 22$ have Lagrange multipliers $\lambda_3 = 36$ and $\lambda_4 = -6$ respectively. Thus, the constraint with the largest Lagrange multiplier from $\mathcal{W}$ has to be removed.

In the second iteration though, the quadratic programming solver starts exhibiting it's unique behavior. The solution of the working graph suggests a new iterate at $x_k = (11, 11)$ as shown on figure 3.12. Clearly there are no constraints blocking the advance of this solution so with $\alpha = 1$ the step-length is preserved and the loop proceeds to the next iteration.

The only way to continue advancing is to remove the last remaining constraint. The dual graph seen in figure 3.13 checks for a positive multiplier ($\lambda_4 = 12$) to be removed from $\mathcal{W}$.

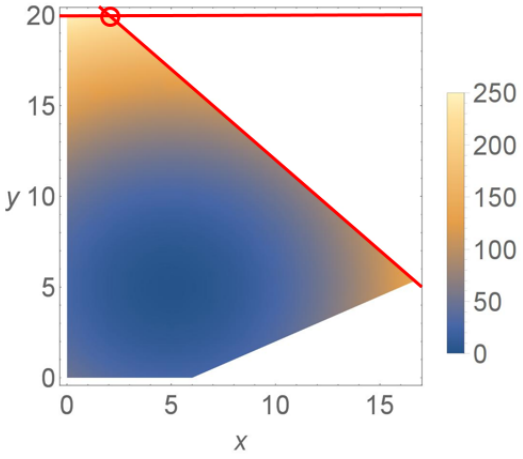

Figure 3.10: This figure shows the initial iterate of the active set solver at $(2, 20)$ and the active set highlighted in red when solving Equation 3.2.

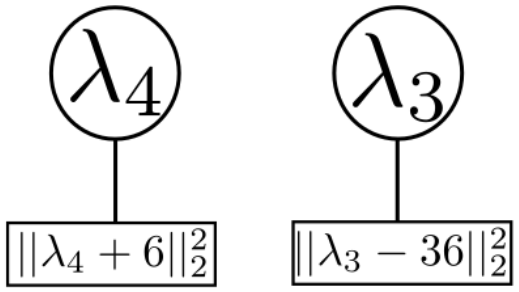

Figure 3.11: This is the dual graph solved during the first iteration of the active set solver when processing Equation 3.2.

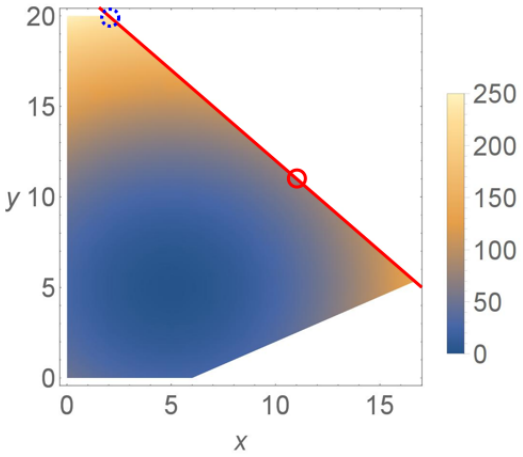

Figure 3.12: The active set solver will have the iterate circled in red when solving Equation 3.2. The active set is highlighted in red at $(11, 11)$. The previous iterate is circled with blue dashes at $(2, 20)$.

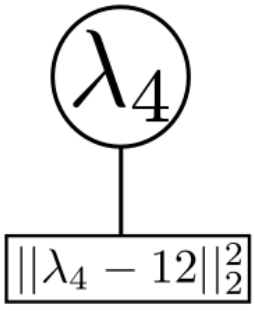

Figure 3.13: This shows the dual graph solved by the active set solver in the third iteration when solving Equation 3.2.

In the fourth iteration of the algorithm the iterate is translated to the center of the quadratic at $(5, 5)$ where it has reached the minimum. In the fifth and final iteration

the algorithm will converge because there are no remaining constraints in the active set which is a terminating condition.

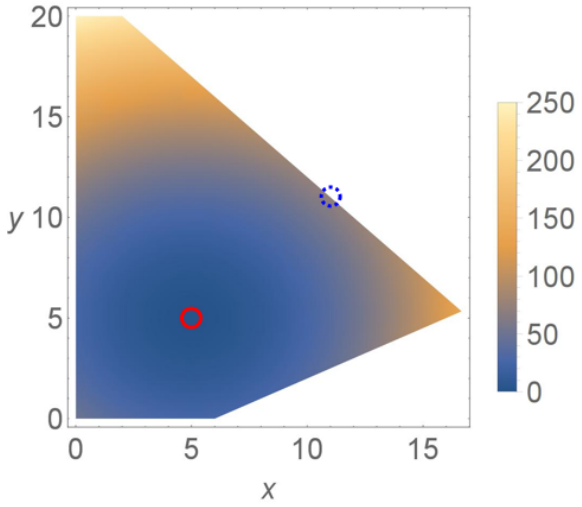

Figure 3.14: In the fourth iteration of solving Equation 3.2, the iterate has moved from the blue dashed circle at $(11, 11)$ to $(5, 5)$ indicated by the red circle. There are no constraints in the active set.

3.4 Solving Large Quadratic Programs

As part of the testing the algorithm, parts of the Maros Mészáros Convex Quadratic Programming Test Problem Set were used [8]. Figure 3.15 and figure 3.16 are the working graph and dual graph solved during the 5^{th} iteration of the algorithm respectively while it solves the problem dual2. This problem has 96 variables, one equality constraint, and 192 linear inequality constraints.

Currently, GTSAM actively uses 13 problems from the data set as part of the unit tests for this tool. It also includes 4 disabled unit tests with larger problems that may be use to profile much larger problems.

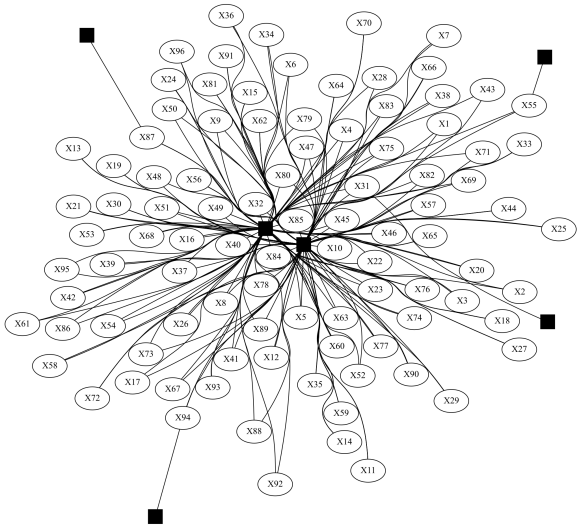

Figure 3.15: This factor graph represents the working graph solved in the 5th iteration of the algorithm while solving the DUAL2 problem of the Maros Meszaros Problem Set.

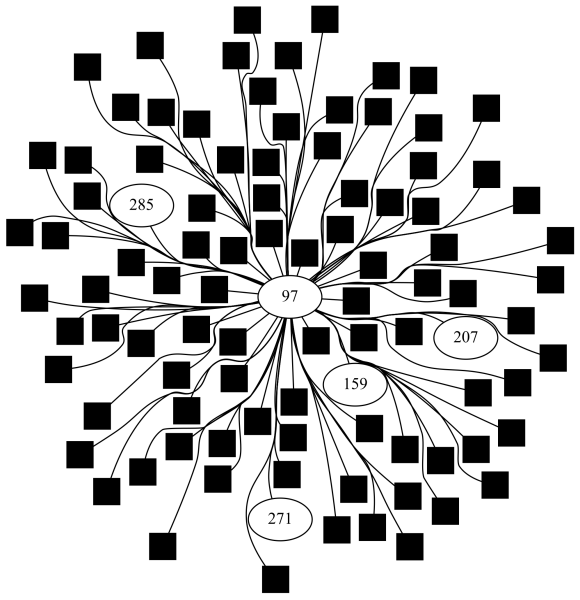

Figure 3.16: This factor graph represents the dual graph solved in the 5th iteration of the algorithm while solving the DUAL2 problem of the Maros Meszaros Problem Set. Each variable node stands for the lagrange multiplier of a constraint.

3.4.1 GTSAM Solving an Equality Constrained Program

$$\begin{aligned} &\min(x-1)^4 + (y-1)^4 s \\ &\text{s.t.} \quad y = (x-1)^3 + 1 \end{aligned} \tag{3.3}$$

For this exercise I assume an initial location $x_0 = (3, 9)$ with corresponding dual $\lambda_0 = 0$. The problem can be visually represented in the figure 3.17. Notice that the initial location is far away from the from the origin. From both the image and the representation of the problem it can be deduced that the minimum is located at $(0, 0)$.

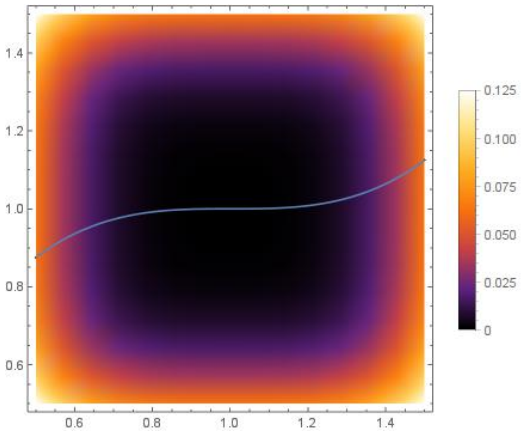

Figure 3.17: This is a visual representation of the problem in Equation 3.3. The background's shade is used to show the cost function while the constraint is shown as a blue line overlaid on top of that.

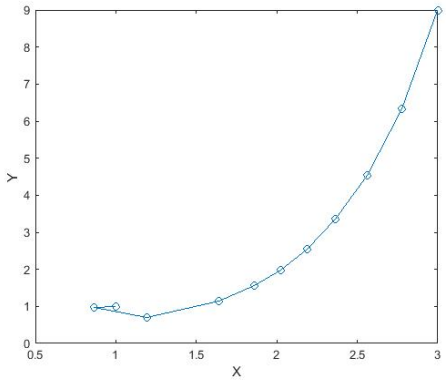

Figure 3.18: This is a graph of the iterate as it goes from it's starting position to the solution of the program. Notice how the steps get shorter as the iterate approaches the solution.

You can see in figure 3.18 how the algorithm converges through its 13 iterations. Notice that it stays very close or exactly on the constraint through most of the process. In the last few iterations the algorithm overshoots slightly but quickly reaches the minimum.

At every iteration a linearization factor graph must be constructed to be solved

by the QP Solver. The objective function of this subproblem can be represented with the factor graph shown in figure 3.19. In that factor graph, the cost function is represented by the black factor while each equality constraint would be a green factor.

Figure 3.19: This a typical factor graph used to represent the cost function of the QP Subproblem at each iteration. The black factor represents the quadratic cost $p^T \nabla^2 \mathcal{L}_k p + p^T \nabla f_k$ while the green factor represents the linearization of the error on the constraint.

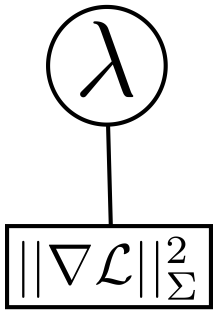

Figure 3.20: During a typical iteration, the algorithm will construct the following this dual graph used to represent the KKT stability condition. The algorithm only uses this graph to calculate the error on the KKT condition since it depends on the QP and the iteration process to compute the actual values of λ.

Finally, during the convergence check the SQP solver has to build a dual graph like the one shown in figure 3.20. In this case the values of the lambda value is given and the factor is use to determine the error on the KKT stationarity condition.

CHAPTER 4
CONCLUSION

As it stands I have presented two implementations of the active set method that are capable of solving linear and quadratic optimization problems in GTSAM. Although both of them have exponential complexity they work well in practice [3]. This is especially true in applications where you have a good initial guess. More importantly, this demonstrates that GTSAM can also be used as a general-purpose constrained optimization framework.

Such a framework should be able to solve nonlinear constrained optimization problems. To this end I have presented a Sequential Quadratic Programming solver based on a Line Search method that can solve some nonlinear constrained programs.

Constrained nonlinear optimization is the most general form of optimization program which can be used to solve robotics problems such as Model Predictive Control and Trajectory Optimization [9]. Although the current implementation is limited in the scope of problems it can solve, I have demonstrated the potential of GTSAM and factor graphs as a constrained optimization framework to both solve and understand these types of problems.

REFERENCES

[1] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," 2012.

[2] D.-N. Ta, "A factor graph approach to estimation and model predictive control on unmanned aerial vehicles,"
International Conference on Unmanned Aircraft Systems (ICUAS), 2014.

[3] J. Nocedal and S. J. Wright, Numerical Optimization, 2nd. New York: Springer, 2006.

[4] D. Hadfield-Menell, C. Lin, R. Chitnis, S. Russell, and P. Abbeel, "Sequential quadratic programming for task plan optimization," in
2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2016.

[5] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range," Automatica, vol. 47, no. 10, pp. 2279–2285, 2011.

[6] A. Cunningham, B. Paluri, and F. Dellaert, "Fully distributed slam using constrained factor graphs," in
Proceedings of the IEEE Conference on Intelligent Robots and Systems, 2010.

[7] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An SQP algorithm for large-scale constrained optimization," SIAM Rev., vol. 47, pp. 99–131, 2005.

[8] I. Maros and C. Mészáros, "A repository of convex quadratic programming problems," Optimization Methods and Software, vol. 11, no. 1-4, pp. 671–681, 1999.

[9] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization.," in Robotics: Science and systems, Citeseer, vol. 9, 2013, pp. 1–10.